Détection hiérarchique multi-classes d'objets dans les images

H. Odabai Fard[1,2] Q.-C. Pham[1] M. Chaouch[1] T. Chateau[2] A. Vacavant[3]

[1] CEA, LIST, Laboratoire Vision & Ingénierie des Contenus, France
[2] Institut Pascal, UMR6602, CNRS, Université de Blaise Pascal, Clermont-Ferrand, France
[3] Image Science for Interventional Techniques, UMR 6284, CNRS, Université d'Auvergne, France

94, F-91191 Gif-sur-Yvette, France
hamidreza.odabaifard@cea.fr

Résumé

Nous présentons une méthode de détection multi-classes qui regroupe différentes classes d'objets dans une hiérarchie pour améliorer le score de détections. Pour parcourir l'arbre, nous proposons d'utiliser un algorithme de recherche efficace permettant de trouver le plus court chemin.

Mots Clef

Détection d'objets, apprentissage multi-classes, détection rapide.

Abstract

We present a multi-class object detection method grouping different classes of objects into a hierarchy to improve classification performance. To traverse the hierarchy, we propose to use an efficient search algorithm to find the shortest path.

Keywords

Object detection, multi-class learning, fast detection

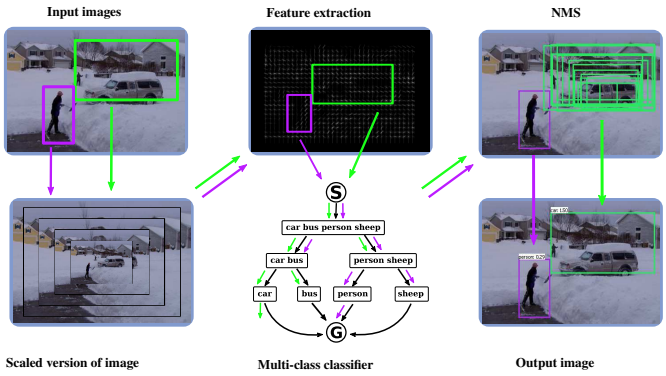

FIGURE 1 – On utilise une approche a fenêtre glissante pour detecter différents objets dans les images. Contrairement a l'approche un-contre-tous, nous n'avons pas besoin d'évaluer tous les filtres de chaque classes, puisque notre systeme essaie de trouver le chemin vers la bonne categorie.

1 Introduction

La détection multi-classes a pour but de localiser des objets (par exemple des piétons, des voitures, des avions,...) dans une image. k étant le nombre de classes à détecter , la méthode naïve consiste à générer k détecteurs, un détecteur pour chaque classe.
L'objectif de la méthode proposée est double :
(1) atteindre une compléxité inférieure à celle de la méthode naïve $\mathcal{O}(k)$.
(2) améliorer les performances de la détection. L'entraînement d'un détecteur avec les données de k classes, peut améliorer les résultats de la détection par la mutualisation d'éléments communs des classes. Par exemple, l'entraînement simultané d'un détecteur pour les classes "voiture" et "bus" peut mieux les distinguer.
Ces objectifs nous conduisent à trois observations : (i) Au lieu de faire une classification entre les classes, on applique un *ranking*. Cette méthode donne souvent de meilleurs résultats qu'une classification [1] ; (ii) en partageant les descripteurs de différentes classes, on est capable de regrouper les exemples de différentes catégories. Ainsi, plus d'exemples sont disponible pour l'entraînement ; (iii) au lieu d'avoir k frontières de décisions, nous souhaitons obtenir plus de frontières de décision, permettant ainsi une séparation plus efficace. Ces réflexions ont mené a un système hiérarchique ou les classes sont des noeuds et elles peuvent être regroupées en *super-classes*, comme illustré en Fig. 1.
Ici, chaque noeud représente un filtre qui extrait un descripteur autour d'un centre d'annotation et auquel il associe un score de similarité. Dans notre cas, ces filtres sont des filtres linéaires. On utilise actuellement comme descripteur le HOG (*histogram of oriented gradients*) [2].
Une hiérarchie donne lieu a plus de k noeuds (filtres). Pour une détection rapide, nous proposons d'utiliser l'algorithme de recherche A* [4] qui ne parcourt que quelques chemins dans l'arbre en évaluant peu de noeuds.

2 Le Détecteur

Pour la détection, nous nous basons sur la méthode classique de détection par fenêtre glissante. Le processus de la détection est illustré dans la Fig. 1. Pour différentes tailles de l'image originale, des descripteurs sont extraits. Notre classifieur multi-classes permet de classifier les différentes régions dans l'image. Un objet donne souvent lieu aux réponses positives. C'est pourquoi une étape de suppressions de ces maximas locaux est appliquée (affiché avec NMS (*non-maxima suppression*) dans la Fig. 1). Finalement, on affiche dans cette figure la réponse du détecteur.

Comme mentionné dans la Sec. 1, nous arrangeons les k classes dans une hiérarchie ou les classes similaires sont groupées ensembles (voir la Sec. 3 pour plus d'information). Chaque noeud dans l'arbre représente un filtre w_i associé à une région autour du centre de l'annotation. La multiplication de la concaténation de tous les filtres de chaque noeud w et les descripteurs de chaque classe y produit le score de cette classe. On utilise alors uniquement les descripteurs des régions définies par les noeuds qui sont sur le chemin de cette classe y. Les descripteurs des autres filtres sont mis a 0. En reprenant l'exemple de la Fig. 1, les filtres utilisés pour calculer le score de la classe 'car' sont : 'car bus person sheep', 'car bus' et 'car'. Dans ce cas, le *label* (la classe) associé a chaque région est déterminé par le maximum des scores de toutes les classes :

$$\hat{y} = \underset{y \in \{1,\dots,k\}}{\arg\max}\ w^T \cdot \Phi(x, y) \tag{1}$$

où $\Phi(x, y)$ est la concaténation des descripteurs associés aux noeuds se retrouvant sur le chemin de la classe y.

Comme mentionné avant, nous utilisons l'algorithme de recherche A* qui permet de trouver le chemin avec la plus grande probabilité sans parcourir nécessairement tous les chemins. L'idée consiste à évaluer les noeuds du haut vers le bas. A chaque noeud, on utilise les scores déjà calculés des filtres se situant plus haut dans la hiérarchie ainsi qu'une estimation (determinée lors de l'entraînement). Cette estimation permet de savoir l'apport maximum qu'on peut avoir en continuant un certain chemin. Ainsi, l'algorithme A* détermine à chaque noeud quel chemin dans l'arbre emprunter pour continuer dans la hiérarchie. Nous esperons éviter le calcul lourd du score pour chaque noeud et être plus rapide que $\mathcal{O}(k)$ où k est le nombre de classes.

3 L'entraînement

Etant donné les annotations des objets, la première étape consiste a trouver une hiérarchie ou les classes du même groupe partagent des descripteurs similaires. Dans ce but, on crée un détecteur pour chaque catégorie similaire à [2]. Après, on crée une matrice de similarité entre les différentes classes en moyennant les scores donnés par un détecteur de classe y_i pour les exemples de la classe y_k [3]. Utilisant cette matrice de similarité, on applique hiérarchiquement le *spectral clustering* pour trouver les classes similaires.

Lors de l'entraînement, on détermine les paramètres de chaque noeud. Il s'agit d'un filtre qui permet d'associer à une région un score et une constante permettant de savoir le maximum que les enfants peuvent apporter. Au lieu de faire une classification pour connaître le label de l'objet, nous appliquons un ranking. La classe correcte est classée plus grande que toutes les autres k classes. Le *background* (la région ne contenant pas d'objets) est traité différemment que les k classes : Il est classifié comme négatif. Donc, un score négatif indique une région background. Ainsi, on mélange la classification et le ranking dans un unique problème d'optimisation :

$$\begin{aligned} \min_{w,\xi \geq 0}\ & \frac{1}{2}\|w\|^2 + \frac{C}{n}\sum_{\xi_i=1}^{n}\xi_i, \\ \forall i, \forall y \in \mathcal{Y}_k \setminus y_i : & \langle w, (\Phi(x_i, y_i) - \Phi(x_i, y_k)\rangle \geq 1 - \xi_i, \\ \forall i, \forall y \in \mathcal{Y}_k : & sgn(y_i)\langle w, \Phi(x_i, y)\rangle \geq 1 - \xi_i. \end{aligned} \tag{2}$$

Ici, n est le nombre total d'exemples, x_i un exemple donné par l'annotation et C un paramètre pour éviter l'*overfitting*. La première contrainte assure un ranking entre les classes si le label appartient à une des k classes. La deuxième contrainte force un score négatif pour les régions du background. Pour résoudre ce problème d'optimisation, nous utilisons l'outil en ligne SVM^{struct} [5]. Finalement, on souhaite apprendre des constantes permettant de connaître le gain maximal de chaque chemin a partir d'un noeud.

4 Conclusion

Nous avons présenté un systeme où on espère améliorer les résultats avec une hiérarchie entre les k classes. Notre approche est automatique : l'algorithme crée un arbre groupant des classes similaires ensemble. Chaque noeud étant un filtre, nous les entraînons avec un apprentissage mélangeant la classification et le ranking. Lors de la détection, on utilise l'algorithme de recherche A* pour trouver le bon chemin. Les expérimentations sont actuellement en cours. Les tests préliminaires sur PASCAL VOC ont montré une première validation de notre approche.

Références

[1] S. S. Bucak, K. Mallapragada P, R. J., and A. K. Jain. Efficient multi-label ranking for multi-class learning : Application to object recognition. In *ICCV*, pages 2098–2105, 2009.

[2] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *CVPR*, pages 886–893, 2005.

[3] G. Griffin and P. Perona. Learning and using taxonomies for fast visual categorization. In *CVPR*, 2008.

[4] B. Raphael P. E. Hart, N. J. Nilsson. A formal basis for the heuristic determination of minimum cost paths. In *IEEE Transactions on Systems, Science, and Cybernetics*, volume SSC-4, pages 100–107, 1968.

[5] I. Tsochantaridis, T. Hofmann, T. Joachims, and Y. Altun. Support vector machine learning for interdependent and structured output spaces. In *ICML*, pages 104–112, 2004.